2025 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) | 979-8-3315-2710-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISLPED65674.2025.11261799

Always-On Sensing in Energy-Harvested Systems via Stochastic Intermittent Computing

Sepehr Tabrizchi[1]*, Mehran Moghadam[2]*, Ali Shafiee Sarvestani[1], Sercan Aygun[3]
M. Hassan Najafi[2], Arman Roohi[1]
[1]University of Illinois Chicago, Chicago, IL, USA
[2]Case Western Reserve University, Cleveland, OH, USA
[3]University of Louisiana at Lafayette, Lafayette, LA, USA
{stabr, ashaf25, aroohi}@uic.edu, {moghadam, mhassan.najafi}@case.edu, sercan.aygun@louisiana.edu
* These authors contributed equally.

***Abstract*—This paper introduces Stochastic Intermittent Computing (STIC), a framework that integrates intermittent computing (ImC) and stochastic computing (SC) to enable always-on sensing in energy-harvested systems. STIC dynamically adjusts computational precision based on available energy, eliminating the need for non-volatile memory checkpointing traditionally used in ImC systems. By adapting precision in real-time, STIC ensures continuous operation even under severe power fluctuations, significantly improving energy efficiency and system resilience. Evaluation results demonstrate that STIC achieves substantial reductions in area, power, and energy consumption owing to the simplicity of SC and its tolerance to aggressive voltage scaling. Evaluations across multiple neural networks and charging traces confirm that STIC enables robust, low-power edge intelligence for resource-constrained environments.**

I. Introduction

Synergizing emerging computing paradigms is a promising approach to leveraging the strengths of different yet complementary techniques targeting similar goals. Energy-constrained systems, such as wearables, health monitoring devices, biomedical implants, and applications deployed in hard-to-reach or maintenance-intensive environments, require power- and energy-efficient algorithms. Two such paradigms have recently gained attention for resource-aware solutions: intermittent computing (ImC) and stochastic computing (SC). ImC offers a reliable computing model under unpredictable or unreliable power conditions. It enables feasible computation through checkpointing mechanisms [1], [2] and non-volatile memory (NVM) management [3], [4] in the absence of a continuous power source or some with possible interruptions. Techniques such as task-based execution and idempotent processing ensure the correctness of execution even across power failures. SC, on the other hand, presents a radically different computing approach by redefining data representation. Instead of conventional binary formats with positional encoding, SC encodes data as random bit-streams with no significant digit. This transformation allows complex arithmetic operations to be reduced to simple logic gates (e.g., multiplication using AND gate or division using a multiplexer-MUX). With its progressive precision, minimal arithmetic cost, and robustness to noise and soft errors, SC is well-suited for energy-constrained systems. Combining ImC and SC forms a synergistic design approach, where SC's error tolerance complements ImC's resilience to power interruptions. SC's low hardware complexity aligns with the lightweight demands of intermittent systems, enabling practical and impactful integration.

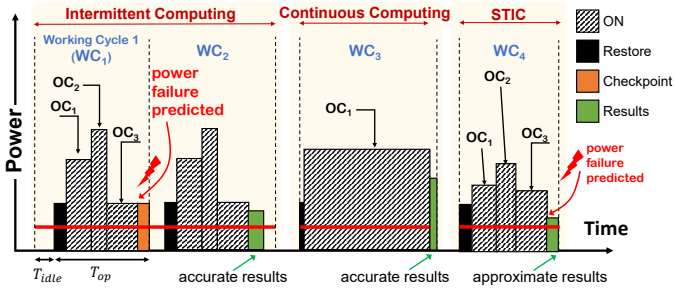

Fig. 1. Three computing paradigms, ImC, CC, and the proposed STIC.

In this work, we propose **STIC** (Stochastic Intermittent Computing), a novel architecture that synergistically integrates the advantages of ImC and SC paradigms within an energy-harvesting environment, as illustrated in Fig. 1. Unlike conventional ImC systems—which operate through multiple distinct operating cycles (OC), each comprising periods of computational operations (T_{op}), idle times (T_{idle}), and mandatory checkpointing intervals—**STIC** fully removes the need for checkpointing by exploiting the intrinsic noise tolerance and progressive precision capabilities of SC. Specifically, checkpointing elimination provides two major advantages: ① enhanced *memory and transmission efficiency*, as intermediate states no longer need to be periodically stored or transmitted; and ② support for *always-on operation*, thus removing interruptions caused by checkpointing and state restoration, and enabling smooth, uninterrupted computations akin to continuous computing (CC). Comprehensive experiments across three distinct energy profiles (CT1–CT3) demonstrate that **STIC** achieves significant area savings and maintains robust computational accuracy, even under aggressive voltage scaling (up to 40% reduction), clearly outperforming both conventional ImC and CC approaches in resilience, adaptability, and overall system efficiency.

This manuscript is structured as follows: Section II provides background on ImC and SC. Section III introduces the proposed **STIC** architecture, detailing its algorithmic and architectural aspects. Section IV presents the validation and evaluation results, and finally, Section V concludes the paper.

II. Background

The increasing demand for energy-efficient and fault-tolerant computing has driven research into alternative

paradigms that can function under stringent energy constraints and in unreliable environments. Two such paradigms, ImC [5] and SC [6], have emerged as promising solutions to these challenges. ImC enables computation in environments with sporadic power availability, while SC leverages probabilistic representations to achieve efficient and low-power computations. This section provides an overview of both paradigms, highlighting their fundamental concepts, prior research efforts, and the associated advantages and drawbacks.

A. Intermittent Computing (ImC)

ImC is a paradigm designed to ensure forward progress in computation, even when power is unreliable or available only in short bursts. This is particularly relevant for energy-harvesting systems that rely on ambient sources such as solar, radio-frequency, or thermoelectric energy, which are inherently unpredictable [7]–[10]. Unlike traditional computing systems that assume a continuous and stable power supply, ImC frameworks enable operation across multiple power cycles by preserving the computational state between power losses. The fundamental principle of ImC is *checkpointing* and *restoration*, where the system periodically saves its state to NVM and restores it upon reactivation. Recent advancements such as task fragmentation, idempotent execution, and adaptive checkpointing further enhance the efficiency of ImC systems.

Several research efforts have advanced ImC. Hibernus [11] was one of the first frameworks to introduce task-based intermittent execution. It employed a reactive checkpointing mechanism that minimized overhead but required significant hardware modifications. Mementos [12] proposed a software-only approach using compiler-assisted checkpointing to detect power failures and store the system state effectively, making it an attractive option for low-power embedded systems. Clank [13] introduced techniques for idempotent execution, ensuring correctness despite unexpected power failures by restructuring computations. This framework aimed to mitigate inconsistencies caused by power loss and improve execution efficiency by reducing redundant operations. QuickRecall [14] focused on optimizing NVM usage to improve performance in energy-harvesting systems, providing an innovative approach to energy-efficient computation through memory-aware task scheduling. Recently, Bamboo [15] proposed a novel checkpointing method that dynamically adjusts based on power availability, significantly reducing latency and energy consumption while improving system responsiveness to fluctuating power conditions. ImC ensures resilience to power failures by enabling computation to progress despite unstable power conditions. It improves energy efficiency, making it ideal for battery-less systems that rely on ambient energy. Additionally, many implementations leverage software-based solutions, reducing hardware overhead. Despite its advantages, ImC comes with some limitations. Frequent state-saving operations can introduce performance penalties due to checkpoint overhead. Excessive writes to NVM can degrade the longevity of storage components, posing challenges in long-term deployments. Ensuring correctness and forward progress in applications requires additional software and architectural support, adding complexity to execution models.

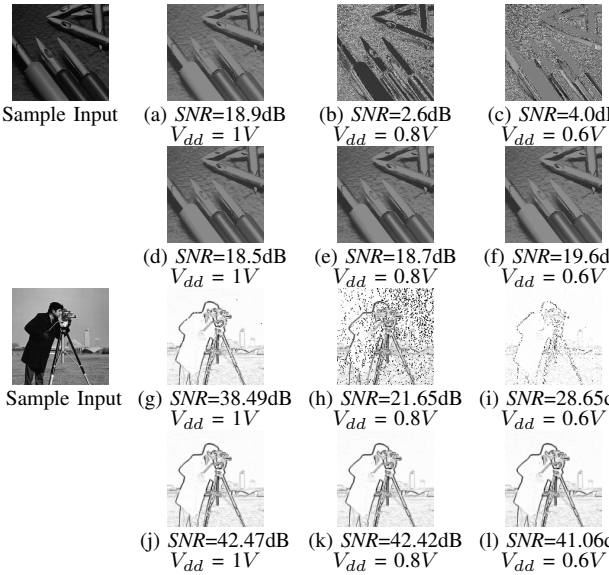

Fig. 2. Effect of voltage scaling on computing systems. **Gamma Correction** circuit [20]: (a)-(c) results of conventional binary computing and (d)-(f) results of SC, **Robert's Edge Detection** circuit [21]: (g)-(i) results of Conventional binary computing and (j)-(l) results of SC.

B. Stochastic Computing (SC)

SC [6], [16] is an emerging computing paradigm that fundamentally departs from conventional binary computing by representing numerical values using randomly distributed bit-streams. Instead of encoding data based on positional binary radix, SC encodes values probabilistically: a bit-stream of length N containing N_1 ones represents the value $X = \frac{N_1}{N}$, where X also corresponds to the probability of observing a '1' in the bit-stream. This probabilistic representation makes SC *highly robust to noise*, i.e., bit-flips, as any bit flip can contribute only the least significant bit error. Multiple bit-flips can statistically cancel out each other, resulting in negligible distortion in the final value. Another key advantage of SC lies in its *hardware simplicity*–Complex arithmetic operations can be implemented using extremely simple logic systems. For instance, multiplication can be implemented with a single `AND` gate while division can be performed using a `MUX` [17]. SC designs have demonstrated 50× to 100× reductions in both hardware area and power costs compared to conventional binary designs across a wide range of applications. The paradigm also benefits from a *progressive precision* property, which enables dynamic control over computation accuracy simply by adjusting the bit-stream length and no change in the architecture. While processing longer bit-streams (e.g., 256 bits) can yield highly accurate results, for many applications, processing shorter bit-streams (e.g., 32, 16 bits) can achieve sufficient accuracy [18], [19].

Another distinguishing feature of SC is its *high tolerance to aggressive voltage scaling*. By reducing supply voltage, the system's power and energy consumption can be significantly reduced with minimal impact on output quality. Aggressive

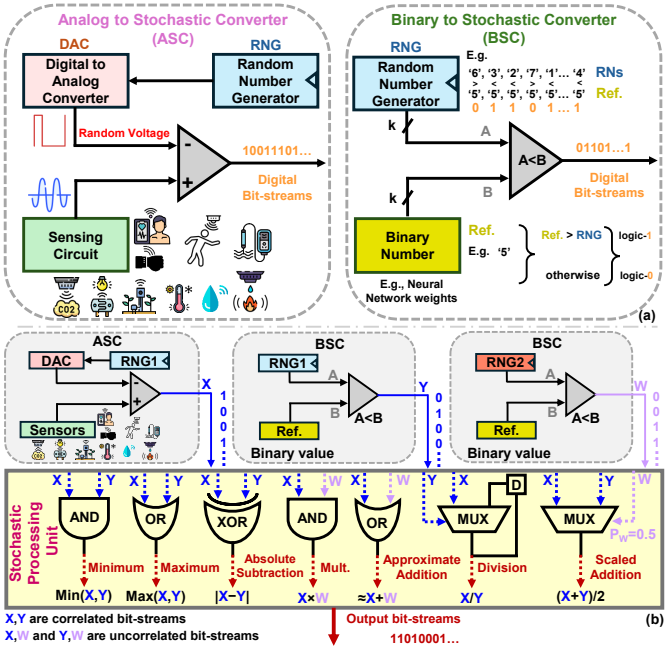

Fig. 3. Computation core of **STIC**, (a) conversion to SC bit-streams, (b) SPU as the processing block implementing arithmetic operations.

voltage scaling in conventional binary circuits often introduces numerous timing errors, resulting in substantial degradation in the computational accuracy. However, in SC circuits, the timing errors induced by voltage scaling tend to statistically cancel each other out [20], resulting in minimal impact on the overall accuracy. Fig. 2 shows output images generated by conventional and SC circuits at different voltage levels. The output quality is quantified with the signal-to-noise ratio (SNR). As shown, even modest voltage reductions cause a significant drop in the output quality of binary circuits, while SC circuits maintain high-quality results even under aggressive voltage scaling (e.g., down to 0.6V). This resilience enables significant energy savings in SC circuits, making them more energy-efficient than their binary counterparts without compromising accuracy. Prior studies showed SC circuits can continue to operate reliably with up to 40% reduction in supply voltage, with little to no degradation in accuracy [20].

Fig. 3 demonstrates two primary components of an SC system: I) *Analog-to-Stochastic Converter* (ASC) or *Binary-to-Stochastic Converter* (BSC), and II) *Stochastic Processing Unit* (SPU). The choice between ASC or BSC depends on the input type (i.e., analog or digital binary, as depicted in Fig. 3(a)). The ASC unit converts analog inputs from a sensing circuit into stochastic bit-streams by comparing the input voltage with random voltages generated using a random number generator (RNG) (e.g., Linear Feedback Shift Register) and a *Digital-to-Analog Converter* (DAC). Similarly, the BSC unit generates stochastic bit-streams by comparing a binary input value with random values from an RNG. Assuming one bit of the bit-stream is generated per cycle, producing an N-bit bit-stream requires N comparison operations over N cycles. The SPU consists of basic logical elements to implement various arithmetic operations, as shown in Fig. 3(b). A crucial factor in SPU functionality

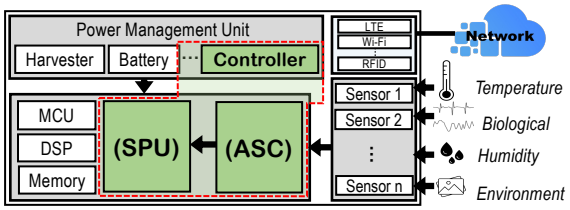

Fig. 4. The proposed **STIC** architecture.

is the correlation between bit-streams [22]. Some operations require independent inputs (e.g., `Multiplication` using `AND` gate), while others require highly correlated inputs (e.g., `Minimum` using `AND` gate). The correlation level is typically managed by sharing the same RNG sources (for correlated bit-streams) or using different RNG sources (for uncorrelated bit-streams) in the conversion units. SC is especially popular for error-resilient applications, machine learning accelerators and low-power sensor networks, where exact computations are not necessary [23].

III. Proposed **STIC** Architecture

SC enables dynamic control of computational precision by adjusting the length of bit-streams. This flexibility allows the system to adapt its precision level based on the available charging rate. Leveraging this property, our proposed framework –**STIC**– can maintain operational effectiveness even under varying energy constraints. This adaptability eliminates the need for NVM, which is typically used in traditional intermittent systems for system backup. By removing NVM, **STIC** significantly reduces power consumption, enhances energy efficiency, and extends the system's operational lifetime, making it particularly well-suited for energy-constrained environments [24]. To utilize this benefit, **STIC** encapsulates three additional components compared to traditional intermittent systems as illustrated in Fig. 4: An *ASC* unit, an *SPU*, and a *Controller*. The SPU, as the core of the system, executes the operations in the stochastic domain.

SC utilizes bit-streams in which all bits have equal significance. As a result, precise representation of N bit binary data requires bit-streams of 2^N bits. For instance, a 4-bit binary calculation requires 2^4-bit bit-streams and 2^4 processing cycles in an SC system. Processing long bit-streams can result in slower performance compared to conventional binary systems. To enhance computational speed, we exploit the inherent parallelism of SC designs and split bit-streams into multiple sub-bit-streams and process them using multiple SPUs in parallel. Since SPUs consist of basic logic gates and are low-cost, multiple copies have minor cost overheads. Fig. 5 demonstrates an example of 4× parallelization, where four stochastic bits are generated simultaneously in each clock cycle. When the computation unit (i.e., SPU) requires correlated bit-streams, a shared RNG source can be used. When independent bit-streams are needed, multiple sets of random numbers can be derived even from a single RNG source, thanks to the recent advancements in SC systems [25], [26]. This ensures minimum hardware overhead when uncorrelated bit-streams are needed for specific operations like multiplication.

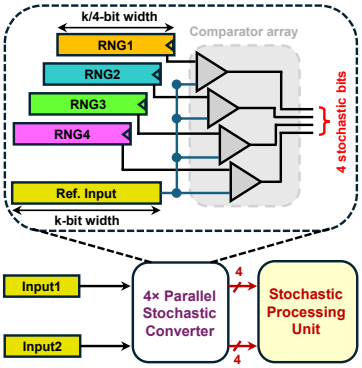

Fig. 5. Example of 4× parallelized structure of **STIC** design by generating simultaneous bit-streams in $N/4$ clock cycles.

TABLE I
MAE (%) OF SC OPERATIONS/CIRCUITS FOR VARIOUS NUMBER OF CYCLES

Bit-stream length, N	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
2-input Multiplication	3.73	1.84	0.92	0.45	0.19	0.09	0.04
3×3 MAC	29.9	15.4	8.03	4.48	1.77	0.89	0.45
3×3 Median Filter [27]	3.08	1.59	0.82	0.45	0.26	0.08	0.04
Gamma Correction [16]	7.75	5.36	3.53	2.39	1.67	1.18	0.85
Robert's Edge Detection [28]	3.85	1.91	0.95	0.47	0.23	0.11	0.05

Algorithm 1 outlines the procedure executed by the SPU. The parameter p in the first line represents the number of parallel SPUs utilized, where $p \in \{2^k \mid k \in \mathbb{N}\}$. The second line defines the set of valid bit-stream lengths, chosen as powers of two corresponding to equivalent binary bit-widths. The core SC procedure begins at line 3, where computations are performed in the stochastic domain. At line 5, a new data is fetched, followed by computational operations that run for up to $256/p$ cycles (line 6). For most applications, a bit-stream length of 256 (equivalent to 8-bit precision in binary) provides sufficient accuracy. When the current clock cycle matches one of the predefined valid lengths, the SPU interacts with the *Controller* by invoking the CHECK_CHARGING_RATE procedure (line 9).

Within the CHECK_CHARGING_RATE procedure, the system assesses the circuit's charging rate and returns a *precision level* between 0 (lowest) and 4 (highest), based on some predefined thresholds Th_1 to Th_4. These thresholds are determined by balancing the required computation power and a designated margin (ϵ). The primary goal is to maintain the battery level at the maximum value by charging the battery at least with this margin. Thus, the battery is only utilized when the charging rate drops below the lowest threshold (Th_1), indicating that even low-resolution (16-bit) computations cannot be sustained from the charging source. In the SC_COMPUTATION procedure, once the appropriate valid index is determined (line 9), a comparison is made against the current clock cycle (line 10). If the clock cycle is less than this valid length, it indicates that sufficient energy is available, and computation can proceed. However, if the clock value meets or exceeds the valid index, it signals limited available energy. In this case, the loop exits at line 11, and the system moves on to the next input data.

IV. VALIDATION AND EVALUATION

To evaluate the performance of **STIC** compared to traditional ImC, we first show how the accuracy of SC improves

Algorithm 1 SC_computation and check_charging_rate

```
1: p ← 4                                   ▷ Number of parallel cores for one input (e.g. 4)
2: valid_length ← {16, 32, 64, 128, 256}/p   ▷ Divide each valid length by p
3: procedure SC_COMPUTATION
4:     while true do
5:         new_data ← READ_NEW_DATA
6:         for clock ← 0 to 256/p do
7:             DO_COMPUTATION_IN_PARALLEL
8:             if clock == valid_length then
9:                 valid_index ← CHECK_CHARGING_RATE   ▷ Find the best precision based on charging rate
10:                if clock ≥ valid_length[valid_index] then
11:                    break                ▷ Exit the for loop
12:                else
13:                    COMPUTE(new_data) ▷ Do the computation for another cycle

14: procedure CHECK_CHARGING_RATE
15:     charging_rate ←READ_CHARGE_RATE()
16:     if charging_rate > Th4 then                ▷ 256 bits
17:         return 4
18:     if Th3 < charging_rate < Th4 then          ▷ 128 bits
19:         return 3
20:     if Th2 < charging_rate < Th3 then          ▷ 64 bits
21:         return 2
22:     if charging_rate < Th2 then                ▷ 32 bits
23:         return 1
24:     return 0                                   ▷ 16 bits
```

over time. Table I reports the Mean Absolute Error (MAE) of some selected SC functions for different processing cycles (i.e., bit-stream lengths, N). We run each operation 10,000 times, each time with a random data set in the [0,1] interval. As can be seen, for all operations, the MAE reduces as the bit-stream length increases. Table II compares the hardware costs of some SC designs and their traditional binary counterpart. As reported, the SC designs achieve significantly lower hardware costs than their traditional counterpart. Fig. 6 shows the visual outputs of running a noise removal filter and an edge detection circuit using traditional ImC and the proposed **STIC** for different run-times. As Figs. 6(a),(b),(h), and (I) show, the results of the ImC are incomplete at 25% and 50% run-time with low structural similarity (SSIM) values. The processing must continue during the next active period to produce an acceptable output. However, **STIC** produces an output all the time with improving quality over time. For example, the SSIM values increase from 0.38 to 0.70, 0.92, and 0.99 for 25%, 50%, 75%, and 100% run-time for the noise removal filter and from 0.55 to 0.95, 0.98, and 0.99 for 2%, 12%, 25%, and 100% run-time for the edge detection circuit validating the quality improvement over time.

Next, we evaluate the performance of **STIC** using various neural network architectures under different precision levels. For this analysis, we implemented a single SPU core and a BSC in 65 nm technology using HSPICE and PTM library, and measured the energy consumption per MAC operation.

TABLE II
HARDWARE COST COMPARISON OF THE COMPUTATION UNIT

Operations	**Computation Approach**	**Area** (μm^2)	**Power@Max freq.**(mW)	**CPL*** (ns)	**Energy** (pJ)
3×3 MAC	CC	6378	6.60	1.30	8.58
	SC	**193**	**0.19**	**0.63**	30.6
3×3 Median Filter	CC	2167	1.03	2.10	2.16
	SC	**79**	**0.05**	**0.39**	5.19
Gamma Correction	CC	1153	0.82	1.10	0.90
	SC	**76**	**0.10**	**0.63**	16.1
Robert's Edge Detection	CC	574	1.16	0.78	0.91
	SC	**14**	**0.02**	**0.30**	1.54

*: Critical Path Latency||Synthesized using Synopsys Design Compiler with 45nm library.

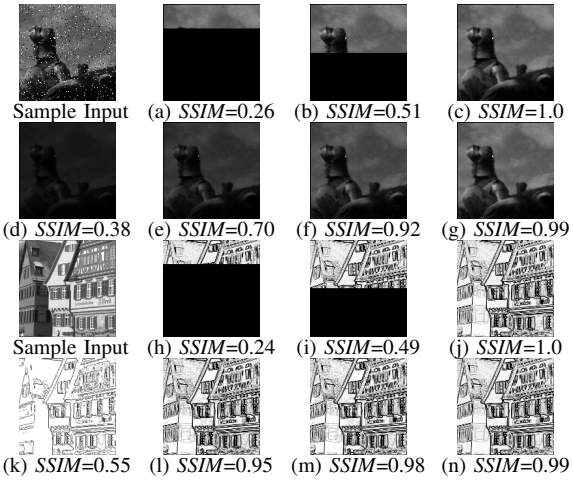

Fig. 6. **Noise removal median filter**; output with ImC: (a) 25%, (b) 50%, (c) 100% runtime, and **STIC**: (d) 25%, (e) 50%, (f) 75%, (g) 100% runtime. **Edge Detection**; ImC: (h) 25%, (i) 50%, (j) 100% runtime, and output with **STIC**: (k) 2%, (l) 12%, (m) 25%, (n) 100% runtime.

We then conducted 10 separate inference tasks across multiple neural network architectures. The results, presented in Fig. 7, indicate that ShuffleNet consistently outperforms other networks, exhibiting the lowest energy consumption across all evaluated precisions.

To ensure a fair comparison, we benchmarked the proposed **STIC** at 8-bit precision (equivalent to 256-bit bit-streams) against several state-of-the-art in-memory computing (IMC) architectures employing different technologies at the same precision level, as shown in Fig. 8. Initially, our design appears to lag behind STT-MRAM and SOT-MRAM architectures when the overhead of the BSC unit is included (**STIC** with BSC). However, in practical scenarios where many SPUs operate in parallel, the BSC overhead becomes negligible, as its RNG will be shared among many SPUs. By leveraging this architectural advantage and eliminating the BSC overhead (represented by the dark blue bar in Fig. 8), our **STIC** design, equipped with an 8-bit precision MAC achieves superior energy efficiency compared to alternative technologies. Once again, ShuffleNet stands out as the most energy-efficient network, reinforcing our selection of this architecture as the baseline for our subsequent analyses.

Fig. 9 provides a detailed breakdown of the proposed **STIC** energy consumption across individual network layers under varying precision levels. As illustrated, fully connected (FC) layers generally exhibit lower energy consumption, whereas Shuffle units exhibit higher power demands

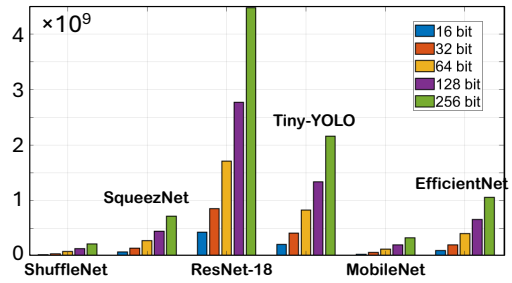

Fig. 7. **STIC** energy consumption for various networks and precisions.

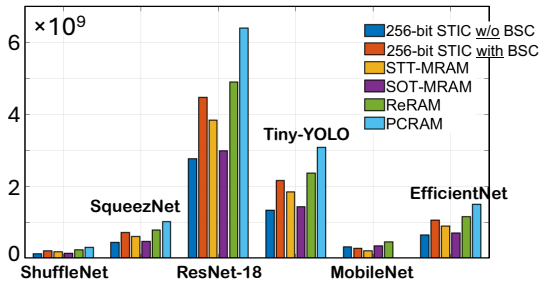

Fig. 8. Energy consumption of **STIC** versus ImCs using various beyond CMOS technologies.

relative to other layer types. It should be noted that these simulations do not include network-specific optimizations. Future energy efficiency improvements can be achieved by exploiting the compatibility of SC with bitwise operations, such as employing `XOR`-based channel shuffling to further reduce computational overhead.

The key results of our evaluation are summarized in Table III and Fig. 10. Initially, we normalized the MAC energy consumption of **STIC** with respect to the MAC energy specified in the STM32F107vc datasheet. Subsequently, we utilized the simulation tools proposed in [29] to evaluate three different architectures: a normal (conventional) microcontroller-based system, a reactive intermittent system, and the proposed **STIC**, under three distinct Charging Traces (CT1-CT3). The continuous computing system (CC) refers to a conventional microcontroller setup without additional mechanisms for power management, performing computations continuously whenever sufficient energy is available until the energy drops below the shutdown threshold. The ImC system, using the reactive method, employs a microcontroller augmented with NVM. This architecture integrates an early-stop mechanism that proactively transitions the microcontroller into a sleep state before power failure occurs, thus reducing backup-related overhead. However, NVM writes are energy-intensive, significantly increasing energy consumption.

In contrast, the proposed **STIC** does not require NVM backup; thus, its state machine excludes `Sleep`, `Load`, and `Store` operations. In Table III, the number of sensing cycles corresponds to a complete operational cycle involving sensing, computing, and data transmission. Furthermore, the three charging traces considered represent normal (CT1), favorable (CT2), and severe (CT3) charging conditions. The results demonstrate that even under the harsh conditions of CT3, **STIC** successfully sustains operation and continues computations efficiently (# `Sleep` in the table). Additionally, the

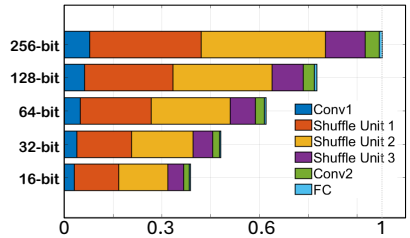

Fig. 9. MAC energy consumption breakdown per layer for ShuffleNet-V2 regarding different precisions in **STIC**.

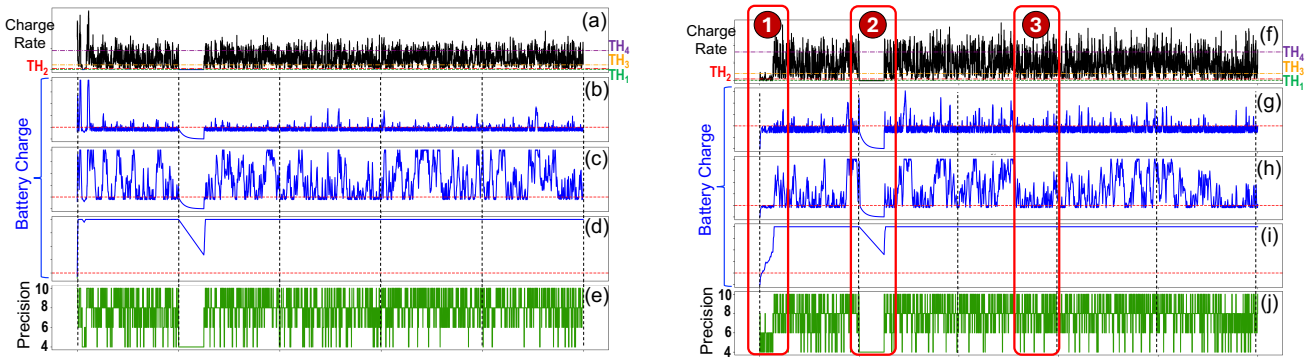

Fig. 10. (a, f) Charging rate and battery level of different designs; (b, g) continuous computing system (CC); (c, h) reactive intermittent (ImC); and (d, i) **STIC** , which offers an always-on scheme. (e, j) Transition in **STIC** due to precision adjustment.

behavior of all three architectures under CT2 (left) and CT3 (right) conditions is visually depicted in Fig. 10. In this figure, subplots (a) and (f) illustrate the received charging rates; (b) and (g) show the energy level of CC; (c) and (h) depict the energy levels ImC; (d) and (i) show the energy profile of **STIC**; and (e) and (j) illustrate the computation precision of **STIC**. As observed, even under severely constrained energy conditions, **STIC** remains operational and consistently maintains superior energy efficiency compared to the other architectures. For instance, in (1), **STIC** charges the battery to its maximum capacity significantly faster than the other systems, as highlighted in Fig. 10(i). In another scenario, shown in (2), when the charging rate drops to zero, **STIC** responds by reducing its computational precision to the minimum level (as shown in subplot (j)), thus conserving energy and continuing operation, whereas the other systems cease operation entirely. A further demonstration of the efficiency of **STIC** is illustrated in (3), where the battery level of the proposed system consistently remains near-maximum capacity, in contrast to significant fluctuations seen in the other systems.

To thoroughly evaluate **STIC**'s adaptive precision capabilities under varying energy conditions, we performed detailed simulations across four SC bit-stream lengths, 16, 64, 256, and 1024 bits, which correspond to binary precisions of approximately 4-bit, 6-bit, 8-bit, and 10-bit, respectively. These simulations used three distinct charging traces (CT1–CT3) representing *normal*, *favorable*, and *constrained* energy scenarios. Prior studies have established that 8-bit (256-bit streams) and 10-bit (1024-bit streams) precision levels are typically sufficient for achieving accuracy comparable to standard 32-bit floating-point arithmetic in neural network tasks [30]. Our simulation results demonstrate **STIC**'s intelligent and power-aware precision adjustment. Under standard conditions (CT1), **STIC** predominantly utilizes 256-bit streams (equivalent to 8-bit precision) for 64% of computations, achieving a balanced trade-off between accuracy and energy efficiency, while strategically allocating 12% of computations to higher-precision 1024-bit streams. When energy availability improves (CT2), **STIC** adaptively increases the proportion of computations using the more precise 1024-bit streams to approximately 33%, significantly enhancing computational accuracy during

TABLE III
COMPARISON BETWEEN CONTINUOUS COMPUTING, IMC BASED ON REACTIVE, AND THE PROPOSED METHOD

	Charging Trace	Standby	Sleep	Sense	Compute	Transmit	Load	Store
CC*	CT1	898	4407	27	2041	26	898	898
ImC**	CT1	108	1715	64	1155	63	106	220
STIC	CT1	10	N/A	284	284	284	N/A	N/A
CC	CT2	872	4186	52	1775	51	872	872
ImC	CT2	75	2955	147	2329	156	74	184
STIC	CT2	8	N/A	286	286	286	N/A	N/A
CC	CT3	1530	6422	28	2380	33	1530	1530
ImC	CT3	90	1073	36	686	45	83	143
STIC	CT3	19	N/A	275	275	275	N/A	N/A

∗CC or continuous computing system, and the target ∗∗ImC here is interactive intermittent computing, which shows better performance compared to other ImC designs [29].

favorable conditions. Under constrained energy conditions (CT3), **STIC** robustly adjusts by increasing the use of shorter 16-bit and 64-bit streams to 10% and 17%, respectively, to maintain critical operations with minimal energy consumption. Notably, even under CT3's generally limited conditions, **STIC** intelligently exploits brief improvements in charging rates to opportunistically increase 1024-bit streams usage above the baseline observed in CT1. This sophisticated real-time adaptive behavior underscores **STIC**'s effectiveness in dynamically optimizing the accuracy-energy balance, ensuring reliable and continuous operation across diverse energy availability scenarios.

V. CONCLUSIONS

We presented **STIC**, a novel framework that integrates stochastic computing (SC) and intermittent computing (ImC) for always-on sensing in energy-harvested systems. By dynamically adjusting computational precision based on available energy, **STIC** eliminates energy-intensive NVM operations while maintaining continuous operation. ShuffleNet implemented with **STIC** demonstrates superior energy efficiency across multiple neural network architectures. Tests under three distinct energy scenarios confirm **STIC**'s resilience, maintaining operation even when conventional systems fail by adapting precision between 16-256 bits. This adaptive architecture enables rapid battery recharging and continuous functioning in severely power-constrained environments, making it ideal for energy-limited edge applications requiring persistent sensing.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grant No. 2504839, 2447566, 2019511, 2339701, and a generous gift from NVIDIA.

REFERENCES

[1] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, "Fast and energy-efficient state checkpointing for intermittent computing," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 6, Sep. 2020.

[2] A. Roohi and R. F. DeMara, "Nv-clustering: Normally-off computing using non-volatile datapaths," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 949–959, 2018.

[3] A. Bhattacharyya, A. Somashekhar, and J. S. Miguel, "Nvmr: non-volatile memory renaming for intermittent computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–13.

[4] S. Tabrizchi, S. Angizi, and A. Roohi, "Diac: Design exploration of intermittent-aware computing realizing batteryless systems," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[5] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent Computing: Challenges and Opportunities," in *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. S. Lerner, R. Bodík, and S. Krishnamurthi, Eds., vol. 71. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, pp. 8:1–8:14.

[6] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2018.

[7] S. Umesh and S. Mittal, "A survey of techniques for intermittent computing," *Journal of Systems Architecture*, vol. 112, p. 101859, 2021.

[8] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 199–213.

[9] N. Taheri, S. Tabrizchi, and A. Roohi, "Intermittent-aware design exploration of systolic array using various non-volatile memory: A comparative study," *Micromachines*, vol. 15, no. 3, p. 343, 2024.

[10] A. Roohi and R. F. DeMara, "Irc cross-layer design exploration of intermittent robust computation units for iots," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 354–359.

[11] P. D. F. Hohlfeld and L. Benini, "Hibernus: Efficient energy management for intermittent systems," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, pp. 1–24, 2014.

[12] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, pp. 159–170, 2011.

[13] B. Lucia and B. Ransford, "Clank: Software-only checkpointing for intermittent systems," in *Proceedings of the ACM International Conference on Embedded Systems*, 2018, pp. 1–12.

[14] M. Andersen and M. B. Taylor, "Quickrecall: A hardware-based approach to energy-aware computing," in *IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 35–40.

[15] Y. Xu and J. Torrellas, "Bamboo: Low-latency checkpointing for intermittent computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1803–1816, 2021.

[16] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *Computers, IEEE Trans. on*, vol. 60, no. 1, pp. 93–105, Jan 2011.

[17] M. S. Moghadam, S. Aygun, S. Asadi, and M. H. Najafi, "Low-cost and highly-efficient bit-stream generator for stochastic computing division," *IEEE Transactions on Nanotechnology*, vol. 23, pp. 195–202, 2024.

[18] M. Hassan Najafi, D. Jenson, D. J. Lilja, and M. Riedel, "Performing Stochastic Computation Deterministically," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2925–2938, Dec 2019.

[19] S. Aygun, E. O. Gunes, and C. De Vleeschouwer, "Efficient and robust bitstream processing in binarised neural networks," *Electronics Letters*, vol. 57, no. 5, pp. 219–222, 2021.

[20] A. Alaghi, W.-T. J. Chan, J. P. Hayes, A. B. Kahng, and J. Li, "Trading accuracy for energy in stochastic circuit design," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, Apr. 2017.

[21] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin, "Architecture considerations for stochastic computing accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2277–2289, 2018.

[22] S. Aygun, M. H. Najafi, M. Imani, and E. O. Gunes, "Agile simulation of stochastic computing image processing with contingency tables," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3474–3478, 2023.

[23] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 2809–2824, 2021.

[24] D. Alsadie, "Efficient task offloading strategy for energy-constrained edge computing environments: A hybrid optimization approach," *IEEE Access*, vol. 12, pp. 85 089–85 102, 2024.

[25] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 361–366.

[26] M. S. Moghadam, S. Aygun, M. R. Alam, and M. H. Najafi, "P2lsg: Powers-of-2 low-discrepancy sequence generator for stochastic computing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 38–45.

[27] M. H. Najafi *et al.*, "Low-cost sorting network circuits using unary processing," *IEEE TVLSI*, vol. 26, no. 8, pp. 1471–1480, 2018.

[28] A. Alaghi, C. Li, and J. Hayes, "Stochastic circuits for real-time image-processing applications," in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, May 2013, pp. 1–6.

[29] S. Tabrizchi, N. Taheri, J. Feng, N. Sehatbakhsh, D. Z. Pan, and A. Roohi, "Src: Sustainable reactive computing for battery-free edge intelligence," in *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2024, pp. 93–98.

[30] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 2809–2824, 2020.